UNIVERSIDAD AUTÓNOMA DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

Recomendación de *Point-of-Interest* mediante un modelo gravitacional

Autor: Grace Katherine Taza Castañeda
Tutor: Fernando Díez Rubio

JUNIO 2021

Recomendación de *Point-of-Interest* mediante un modelo gravitacional

AUTOR: Grace Katherine Taza Castañeda
TUTOR: Fernando Díez Rubio

Dpto. Ingeniería Informática
Escuela Politécnica Superior
Universidad Autónoma de Madrid
junio de 2021

Resumen

El objetivo de los sistemas de recomendación es ofrecer productos o servicios personalizados acorde con los intereses de un usuario. En los últimos años, se han desarrollado un creciente número de técnicas para diversas aplicaciones. Paralelamente, los avances en las Tecnologías de información y comunicación han introducido mejoras en las redes sociales permitiendo a sus usuarios compartir su ubicación geográfica. En este tipo de redes sociales basadas en la ubicación, los usuarios pueden compartir los lugares que han visitado, información y su opinión acerca de ellos, además de establecer conexiones con otros usuarios. Las características y el contexto que intervienen en la recomendación difieren de los aspectos a tener en cuenta en los sistemas de recomendaciones tradicionales, lo que ha convertido este dominio en una rama de la investigación de gran interés y actualmente en auge.

Con este trabajo de fin de grado (TFG) se pretende conocer en profundidad un modelo novedoso de recomendación de rutas turísticas basado en la gravedad conocido como LORE [4]. Los resultados experimentales que Zhang et al. consiguen muestran una mayor calidad en las recomendaciones en comparación con otros sistemas de recomendación del estado del arte.

El modelo gravitatorio propuesto cuantifica el impacto que generan los lugares visitados sobre otros lugares nuevos no visitados simulando la fuerza de atracción que existe entre los ambos. Esto se realiza mediante la combinación de las influencias sociales, geográficas y la popularidad.

Examinaremos las dificultades de la reproducción de modelos de recomendación centrándonos en LORE y evaluaremos su efectividad en una región centralizada, la Comunidad de Madrid. Finalmente, los resultados se discuten bajo el contexto de los retos a los que la investigación de sistemas de recomendación se enfrenta y, proponemos una solución al trabajo desarrollado en este TFG como un posible proyecto futuro.

Palabras clave

Sistemas de recomendación, modelo gravitatorio, puntos de interés, reproducibilidad

Abstract

Recommendation systems aim to provide users customized products and services that meet their interests. In recent year, an increasing number of recommendation techniques have been developed with applications in different domains. At the same time, progress in Information and communications technology have enhanced social networks allowing users to share their geographic location. Users on these types of location based social networks are able to share places they have visited, information related with the places and reviews, as well as, build connexions with other users. The attributes and context involved in the recommendation differ from those considered in traditional recommendation systems making this domain a promising field in research and currently in expansion.

In his Bachelor thesis we aim to gain insight into a novelty gravity-based recommendation route model known as LORE [4]. The experimental results obtained by Zhang et al. show a major recommendation quality in comparison with other state of art recommenders.

The gravity model proposed quantifies the impact of visited places over new unvisited places, mimicking the present attraction force between both places. This task is carried out by joining social, geographic and popularity influences.

We analyse the difficulties involved in reproducibility of recommendation models focusing on LORE's algorithm. Lastly, we evaluate the effectiveness of the model using a dataset centralized around a specific region, Madrid. The results obtained are fully discussed in the context of reproducibility challenges faced by recommender-system research and furthermore, we propose a solution to this work to develop in a future project.

Keywords

Recommender systems, gravity model, points of interest, reproducibility

Agradecimientos

Quisiera agradecer a todas las personas que me han apoyado y prestado ayuda para llevar a cabo este Trabajo de Fin de Grado.

En especial quería dar las gracias a la persona más importante de mi vida, mi madre. Su apoyo incondicional en los buenos y malos momentos, los recursos, ánimos y cariño que me ha aportado a lo largo de estos años en el grado me han hecho crecer como persona y es lo que día a día me da fuerza para seguir adelante. A mi hermana, por su ayuda cuando más lo necesitaba y creer siempre en mí.

Por supuesto, también le agradezco a mi tutor, Fernando Diez, por acogerme, su supervisión constante, apoyo y la oportunidad de realizar este proyecto.

Este trabajo se ha desarrollado en el marco del proyecto "América en Madrid. Patrimonios interconectados e impacto turístico en la Comunidad de Madrid" (AmerMad-CM), financiado por la Comunidad de Madrid (ref. H2019/HUM-5694).

INDICE DE CONTENIDOS

INDICE DE FIGURAS

INDICE DE TABLAS

1 Introducción

El campo de investigación de los sistemas de recomendación ha experimentado una inmensa expansión y avance durante la última década en diversos dominios, como el turismo, donde se ha empleado como herramienta de soporte para la planificación de viajes y rutas, entre otros servicios. Asimismo, el gran desarrollo de los sistemas de geolocalización presentes en dispositivos móviles en el día a día han facilitado la creación de las redes sociales basadas en servicios de localización (en inglés *Location-Based Social Network,* con siglas LBSN) como Foursquare [23], Yelp [25] o Gowalla [24]. En este tipo de plataformas los usuarios pueden relacionarse con otros usuarios y compartir lugares que han visitado, su geolocalización y otros aspectos relacionados como fotos, consejos, reseñas, una puntuación, que reflejan sus preferencias e intereses [3]. Estos servicios tratan de mejorar la experiencia de los usuarios con nuevas sugerencias de puntos de interés (en inglés *Points of Interest,* con siglas POI) con el objetivo de satisfacer sus intereses y preferencias a partir de su propio historial de visitas. El creciente volumen de datos generados por usuarios a diario cada vez que realizan una visita, actividad denominada como *check-in* en un punto, ha abierto la puerta a un incremento en la demanda de diseños de recomendaciones de POI eficaces y de mayor calidad.

1.1 Motivación

Este trabajo es consecuencia de distintas circunstancias e inquietudes personales. En primer lugar, la principal motivación de este es formativa pues tenía interés en realizar una TFG experimental y relacionado con el manejo intensivo de datos, a influencia de las prácticas realizadas durante el curso.

En segundo lugar y en línea con lo mencionado más arriba, el desarrollo de tecnologías de recomendación en el contexto turístico es cada vez mayor. En este sentido surgió la oportunidad de aproximarme a este problema mediante el estudio de reproducibilidad del algoritmo LORE (del inglés **LO**cation **RE**commender) [4] para la recomendación de puntos de interés. La propuesta surgió a partir de un proyecto de la Comunidad de Madrid en la que se pretende recomendar lugares de ocio y entretenimiento como museos, parques, restaurantes, relacionados con la cultura americana con el objetivo

de que tanto residentes como viajeros nacionales o extranjeros puedan explorar la oferta cultural y gastronómica que Madrid ofrece sobre estas regiones.

El proyecto mencionado se encontraba en proceso de investigación sobre la posibilidad de aplicar un modelo gravitacional como técnica de recomendación, derivado de la reconocida Ley de Gravitación Universal de Newton. En esta escena aparece LORE, un sistema híbrido que propone un nuevo enfoque de recomendación de POI basado en:

i. Minería de patrones secuenciales.

ii. Cadenas de Markov aditivas.

iii. El modelo gravitacional mencionado [4].

Con estas técnicas, LORE pretende cubrir los aspectos no considerados en otros modelos de recomendación de POI como la influencia social, la geográfica y los patrones secuenciales de visitas.

1.2 Objetivos

Obtener una recomendación de puntos de interés en una determinada ciudad es un problema real puesto que permite conocer aspectos y lugares que por lo general no tienen tanta promoción. Ciertamente las personas incluyen los avances tecnológicos en su vida cotidiana para la toma de decisiones gracias a la extensa cantidad de información disponible en la WWW (en inglés *World Wide Web*). Desde ir a tomar algo con unos amigos un día normal, como planear un fin de semana. Por este motivo, con el análisis detallado del algoritmo LORE propuesto en este artículo, el objetivo principal que se busca es el de reproducir y explorar alguna posible mejora del método de recomendación de POI, basado en la fuerza gravitacional entre los diversos puntos espaciotemporales, en la ciudad de Madrid con el objetivo de obtener nuevas sugerencias que coincidan con los intereses personales de cada usuario.

A partir de dicho objetivo principal de reproducibilidad, podemos especificar los siguientes objetivos secundarios:

- Investigar sobre sistemas de recomendación en el sector turístico.
- Ampliar conocimientos sobre sistemas de recomendación de POI. Qué son, características, aspectos que intervienen en la recomendación, técnicas que se utilizan y dificultades presentes en la creación de nuevos modelos.
- Definir mejoras del sistema de recomendación LORE.

Para la realización del trabajo hemos empleado un *dataset* customizado obtenido de la versión global de Foursquare [23].

1.3 Organización de la memoria

Esta memoria se organiza de la siguiente manera:

- **Introducción**: Se pone en contexto al lector del trabajo a desarrollar sobre el artículo.
- **Estado del arte**: En este capítulo se resume la situación actual de los principales sistemas de recomendación de POI, los tipos y las técnicas empleadas.
- **Experimentación**: Al tratarse de la reproducibilidad de un algoritmo que ya se ha se ha trabajado previamente, en esta sección se explican en detalle los pasos seguidos en la aplicación de dicho algoritmo al *dataset* elegido enfocado a la ciudad de Madrid, los scripts desarrollados, las pruebas realizadas, etc.
- **Resultados**: En este capítulo exponemos los resultados experimentales con el *dataset* obtenido a partir del mundial de Foursquare [23] y evaluamos su eficacia.
- **Conclusiones y trabajo futuro**: Se presentan las conclusiones de la reproducibilidad del algoritmo, así como posibles mejoras futuras.

2 Estado del arte

Los sistemas de recomendación (SR) son agentes personalizados de información que ofrecen sugerencias de ítems más compatibles y con mayor probabilidad a ser utilizados por los usuarios [6, 7]. En un enfoque más general, se pueden definir como: "cualquier sistema que produce recomendaciones individuales como salida o, tiene el efecto de guiar al usuario de una manera personalizada para objetos útiles e interesantes en un gran espacio de opciones disponibles" [5].

A lo largo de la literatura, se puede observar la aplicación de los SR extendidos a diferentes áreas. Por ejemplo, en sistemas de comercio electrónico, las empresas observan nuestras compras para ofrecernos sugerencias de otros productos que nos puedan interesar y así, conseguir aumentar sus ventas. En las redes sociales se intenta analizar nuestros contactos e intereses personales para conectarnos con nuevos amigos y recomendarnos contenido que vamos a consumir con el objetivo de mantenernos conectados a la red. Además, en aplicaciones LBSN se analiza nuestro historial de visitas para recomendarnos nuevos lugares a visitar.

En este capítulo nos centraremos en la aplicación de los SR al turismo. En concreto, enunciaremos detalles básicos de la recomendación de POI que emplean las LBSN, retos a los que se enfrenta y, citaremos una serie de investigaciones sobresalientes con sus diferentes enfoques y tecnologías dirigidas a este dominio [10].

2.1 Recomendación de POI

La recomendación de POI es una de las tareas fundamentales de las LBSN, ayudando a los usuarios a descubrir nuevas localizaciones que les puedan resultar interesantes [8]. Con el fin de intentar simular el asesoramiento facilitado por agencias de viajes tradicionales, estos sistemas en las LBSN proporcionan una asistencia en línea para que los usuarios planifiquen su destino, así como su posible itinerario. Dado que la recomendación de POI es una rama de los SR convencionales, los modelos de los primeros pueden emplear ideas y técnicas de los segundos, como, por ejemplo, el método de filtrado colaborativo. Sin embargo, muchos de los modelos se basan en la matriz de *rating* usuario-item para predecir la preferencia de un usuario [9]. Este enfoque no es, en muchas ocasiones,

conveniente dada a la dispersión en los *check-in* de los usuarios. Precisamente es por el puente que crea este tipo de recomendadores entre el mundo físico y los servicios de redes sociales en línea que aparecen nuevos retos para los SR convencionales. Aquellos aspectos son vistos en [3], [10], [8] y, son los siguientes:

i. **Relaciones complejas.** La ubicación introduce una nueva forma de relación entre usuarios y entre localizaciones y usuarios. En el comportamiento y, por tanto, el movimiento de un usuario influye las relaciones sociales, por ejemplo, un escenario sería cuando se visita un lugar por recomendación de un amigo o cuando se sale con amigos. También dos personas que frecuentan el mismo lugar o viven en una misma ciudad tienen mayor probabilidad de convertirse en amigos. Tales relaciones promueven nuevos retos para crear modelos que aprendan el comportamiento real de un usuario a través de su registro virtual.

ii. **Contexto abundante:** Las preferencias de un usuario abarcan múltiples tipos de intereses que se pueden aprender de su historial de visitas. Pueden depender de la distancia a ciertas regiones de actividad, si es cerca del trabajo o de su casa, y del tiempo, es muy probable que visiten distintos lugares por la mañana que por la noche. Además de la influencia de sus vínculos sociales, las descripciones, opiniones y fotos acerca de los POI.

iii. **Dispersión de datos y comienzo en frío.** El aprendizaje del comportamiento de un usuario no es una tarea trivial. Los lugares visitados por un usuario son normalmente una pequeña porción de todos los POI de la LBSN e incluso pueden no compartir todos los lugares que han visitado. Por lo anterior, los datos para la recomendación no son suficientes para identificar POI similares. Este fenómeno se conoce como dispersión de datos y es uno de los mayores problemas que limitan la calidad de las recomendaciones. Por otro lado, el problema del comienzo en frío aparece cuando el modelo no realiza sugerencias eficaces para nuevos usuarios o usuarios con pocos *check-in* en su historial o, para la recomendación de nuevos POI. Frente a este problema, se necesitan nuevos modelos o modelos híbridos que utilizan diferentes técnicas de recomendación.

2.2 Modelos de Recomendación

En esta sección, introducimos una serie de modelos de recomendación de POI que abarcan algunas de las técnicas de recomendación más populares así como diferentes tipos de información de contexto [10].

2.2.1 Factorización de Matrices

La factorización matricial (FM) [11] es una técnica empleada en el filtrado colaborativo. Se basa en la descomposición de la matriz de ratings usuario-item en el producto escalar de dos matrices de menor dimensión. En concreto, caracteriza tanto a los usuarios como a los POI mediante un vector de factores latentes que se mapean en el espacio latente de dimensión *f*. Por lo tanto, la matriz de *check-in* $\boldsymbol{C} \in \mathbb{R}^{M \times N}$ se descompone en la matriz de usuarios $U \in \mathbb{R}^{M \times \kappa}$ y de POI $L \in \mathbb{R}^{N \times \kappa}$, siendo M, N y κ el número de usuario, de POI y de factores respectivamente. El vector asociado a cada POI *j* y a cada usuario *i* y, por lo tanto, los factores latentes, vienen dados por l_j y u_i. Los elementos de l_j miden el grado de posesión del POI sobre cada factor, mientras de los de u_i, miden el grado de interés de que tiene un usuario sobre POI con los correspondientes factores. La puntuación final de un usuario respecto a un POI se obtiene con el producto escalar resultante $\widehat{C_{ij}} = l_j^T u_i$ y, para evitar el sobreajuste se emplean los términos de regularización λ_1 y λ_2 obteniendo la función del modelo final: $min_{U,L} \|C - UL\|_F^2 + \lambda_1 \|U\|_F^2 + \lambda_2 \|L\|_F^2$.

Entre los modelos de FM existentes podemos mencionar **RankGeoFM** (del inglés ***Rank****ing based* ***Geo****graphical* ***F****actorization* ***M****ethod*)[12], un modelo que aprende las preferencias de los usuarios mediante una función de ranking sobre el historial de POI e incorpora la influencia geográfica introduciendo una matriz latente extra $U^{(2)} \epsilon \mathbb{R}^{M \times \kappa}$. Además, se construye una matriz de influencia geográfica *W* de tamaño N×N, donde $w_{jj'}$ es la probabilidad de que un POI *j* sea visitado dado otro POI vecino visitado *j'*. Por tanto, considerando solo los *k* vecinos más cercanos $\mathcal{N}_k(j)$ de cada POI *j*, se obtiene la puntuación: $y_{ij} = u_i^{(1)} \cdot l_j^{(1)} + u_i^{(2)} \cdot \sum_{j' \in \mathcal{N}_k(j)} w_{jj'} l_{j'}^{(1)}$, con · como producto escalar y, $u_i^{(1)}$ la i-ésima fila de la matriz $U^{(1)}$.

Otro modelo popular en FM es el framework de ***A****ugmented* ***S****quare error based* ***M****atrix* ***F****actorization* (**ASMF**) [13] que consta de dos pasos: i) aprendizaje de los lugares potenciales para un usuario a partir de las visitadas por tres tipos de amigos e, ii) incorporación de los lugares potenciales a un modelo de FM. Este modelo incluye:

i. Influencia social: se observan las localizaciones de amigos sociales en un LSBN, amigos de ubicación, aquellos que han hecho check-in en los mismos POI y, amigos vecinos, aquellos que viven físicamente cerca.
ii. Influencia de categorías: se introduce una matriz de categorías de POI *Q* que indica la preferencia de cada usuario a una categoría.

iii. Influencia geográfica: la distancia física de los POI afecta en la decisión de que usuario lo visite. Un usuario es más propenso a hacer c*heck-in* en POI más cercanos.

2.2.2 Factor de Poisson

El **Modelo de Factor de Poisson** [14] (con siglas **PFM**) es un modelo Probabilístico Factorial (en inglés *Probabilistic Factor Model*). Asume que la matriz de *check-in* usuarios-POI C se descompone en dos matrices de factores latentes (U y L), la matriz de usuarios y de POI, respectivamente. De modo que, se asume que la matriz de *check-in* sigue una distribución de Poisson con la media en el producto escalar de las dos matrices latentes, $C \sim Poisson(UL^T)$.

En primer lugar, tenemos el modelo **MGMPFM** [15] cuyo framework fusiona dos técnicas: **PFM** y *Multi-center Gaussian Model* (**MGM**). El PFM modela la preferencia de un usuario sobre distintos POI. Mientras tanto, el MGM considera la idea de que un usuario tiende a hacer *check-in* alrededor de unos centros y, que la probabilidad de que un usuario visite un POI es inversamente proporcional a la distancia del centro más cercano. De esta forma, MGM incorpora la influencia geográfica existente en el comportamiento de los *check-in* de un usuario distribuidos alrededor de múltiples centros. Se conoce como *centros* a las áreas de mayor actividad de un usuario, es decir, las áreas donde se concentran el mayor número de *check-in.* a el comportamiento de los *check-in* que se distribuyen alrededor de múltiples centros de actividad.

Otro modelo destacado es el geográfico de factorización probabilística **GeoPFM** [16]. La clave de este framework se encuentra en incorporación de la influencia de la geográfica y la preferencia en cuanto a intereses de un usuario. Gracias a este modelo se realizan recomendaciones basados en la predicción del número de check-ins al mismo tiempo que la movilidad de un usuario: introduce un conjunto de regiones latentes que siguen una distribución multinomial que se integran a un PFM para, finalmente, inferir las preferencias de un usuario.

2.2.3 Modelos Híbridos

Los modelos híbridos fusionan dos o más técnicas de recomendación para mejorar el rendimiento y evitar las dificultades presentes de aplicar solo una.

A este grupo pertenece el modelo USG [17] cuyo framework unifica las preferencias de los usuarios, influencia social e influencia geográfica. El origen de su nombre

proviene explícitamente de los aspectos que intervienen en la recomendación: preferencia de **U**suarios, influencia **S**ocial e influencia **G**eográfica. USG aplica un Filtrado Colaborativo (FC) basado en usuario para aprender las preferencias de un usuario a partir de la similitud entre usuarios con *check-in* en los mismos POI. Por otro lado, para deducir influencia social entre dos o más usuarios adopta un FC basado en amistades enfocado en la similitud entre sus amistades en común y sus *check-in*. En cuanto a la influencia geográfica, se obtiene considerando la probabilidad de que el usuario u_i visite el POI l_j, dado su historial de *check-in* L_i: $Pr[l_j \mid L_i] = \prod_{l_y \in L_i} Pr[d(l_j, l_y)]$, con $d(l_j, l_y)$ la distancia entre los POI l_j y l_y.

Finalmente, otros modelos híbridos como los de Chen et al. [20], Cheng et al. [19] y Kurashima et al. [18] tienen en cuenta la influencia de los patrones secuenciales espacio-temporales que se manifiesta en las visitas de usuarios. De este modo, los patrones secuenciales se emplean para conseguir la probabilidad de que un usuario visite un nuevo POI. No obstante, estos modelos no consideran otros aspectos que limitan la efectividad de las recomendaciones, a saber: la influencia espacio-temporal en las secuencias de *check-in*, la influencia social y la popularidad de los POI, además, para el aprendizaje de la influencia secuencial emplean la cadena de Markov de primer orden que solo se fija en el último POI visitado en vez de todo el conjunto de POI visitados.

Como consecuencia de las limitaciones mencionadas se plantea el modelo basado en la idea física de Gravedad, modelo LORE [4], cuyo algoritmo implementaremos en el siguiente capítulo. Este modelo integra un modelo gravitatorio para el aprendizaje de la influencia secuencial valiéndose de las influencias social, espacio-temporal y de popularidad.

3 Experimentación

En este capítulo explicamos en detalle el desarrollo realizado del algoritmo LORE [4] para el caso de uso personalizado de recomendaciones de POI en Madrid.

El modelo de LORE pretende simular la fuerza gravitatoria entre dos POI, uno visitado por el usuario y otro no visitado, por medio de la incorporación de las influencias espacio-temporal, social y de popularidad a través de una distribución de Ley potencial. Las influencias se aprenden modelando el historial de *check-in* de los usuarios. Para encontrar la *fuerza gravitatoria* que ejerce un POI visitado sobre uno no visitado:

i. Determinamos la masa del POI visitado según la frecuencia de *check-in* del usuario.
ii. Deducimos la masa de un POI no visitado en función de la relación de la frecuencia de visitas, *check-in*, que figuran en la red de amigos del usuario y su popularidad en el LBSN. Cada POI gana relevancia a medida que crece su popularidad entre la población.
iii. Comparamos la distancia física entre los dos POI identificando la relación inversamente proporcional de su distancia espacial y diferencia temporal.

Antes de proseguir con el desarrollo, presentamos la notación a utilizar desde este capítulo en adelante en la Tabla 3.1.

Símbolo	Significado
U	Conjunto de todos los usuarios en un LBSN
u	Un usuario cualquiera: $u \in U$
L	Conjunto de todas las localizaciones (o POI) en un LBSN
l	Un POI cualquiera con coordinadas de latitud y longitud: $l \in L$
t	Instante de tiempo de un check-in
t_c	El instante del check-in de un usuario específico
$\langle u, l, t \rangle$	Check-in que describe a un usuario u visitando el POI l en t instante
D	Colección de check-in de todos los usuarios visitando todos los POI
L_u	Secuencia de check-in de u, extraído de D
$l_i \rightarrow l_j$	Subsecuencia bigrama, dos check-in consecutivos, transición de l_i a l_j
$R_{\lvert U \rvert \times \lvert L \rvert}$	Matriz de frecuencias de check-in, extraído de D
$S_{\lvert U \rvert \times \lvert U \rvert}$	Matriz de enlaces sociales
x	Diferencia temporal entre dos instantes de tiempo
y	Distancia espacial de *Haversine* o semiverseno entre dos POI
p_l	Popularidad del POI l
$q_{u,\,l}$	Frecuencia de check-in social por los amigos del usuario u' en el POI l

Tabla 3.1: Notación clave y significado utilizado en el artículo y extraído de [4]

3.1 Dataset

En lo que concierne al *dataset* empleado durante la evaluación del algoritmo, trabajamos con dos alternativas. En una primera fase, decidimos trabajar con el *dataset* de Yelp. Una vez descargado del portal[1] del desarrollador, trabajamos en el filtrado correspondiente a los POI de interés para el proyecto AMERMAD, el cual llegamos a concluir. Este *dataset* personalizado resultó en 835 683 usuarios, 8 068 019 *check-in* y 28 571 *business* (POI), organizados en 14 categorías como: *Active Life-Parks, Restaurants-American (New), Restaurants-American (Old), Arts & Entertainment-Botanical Gardens.*

Sin embargo, a pesar de disponer de una cantidad importante de información muy útil, nos encontramos algunos inconvenientes de difícil solución. Por un lado, no podíamos construir la matriz de frecuencias de *check-in*, la cual es necesaria para realizar pruebas y evaluar el rendimiento del algoritmo reproducido. Por otro lado, tampoco disponíamos de otras estructuras necesarias para implementar el algoritmo, como por ejemplo la secuencia de *check-in* por cada usuario. Esto es a causa de que ninguno de los ficheros *json* incluyen información acerca de cuáles son los usuarios que realizan los *check-in* en los POI y, tampoco existe una relación entre los distintos ficheros que nos ayude a identificarlos. Por esta razón, tuvimos que rehacer esta fase de filtrado del *dataset* original, pero trabajando otro diferente. En la segunda iteración optamos por emplear la versión con POI mundiales de Foursquare [23], una vez nos cercioramos de que podríamos disponer de todas las estructuras y la información necesarias.

La selección de Foursquare [23] como fuente del *dataset* que nos serviremos, parte de la teoría en la que se sustenta y, por ser un *dataset* con POI del mundo real y de las más voluminosas que se encuentran disponibles en línea. Los ficheros, en este caso, se encuentran en formato de texto, mientras que los datos tras el filtrado se almacenan en ficheros csv por comodidad. La estructura de los datos se conserva del *dataset* original que mostramos en la Figura 3.1, resultando en 828 usuarios, 2551 *check-in* y 1027 *venues* (POI) en Madrid, organizados en 21 categorías recogidas en la Tabla 3.2.

[1] https://www.yelp.com/dataset/documentation/main

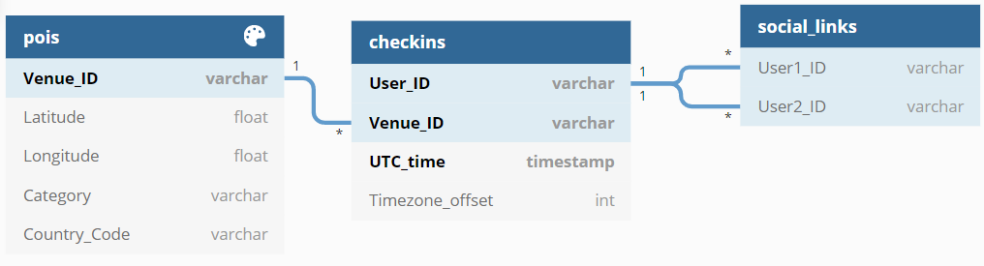


Figura 3.1: Diagrama relacional de la información del dataset personalizado

La fase de filtrado del *dataset* original se descompone en 3 partes significativas:

i. **Obtención de POI de Madrid**: en primer lugar, implementamos dos funciones que leen y filtran todos los *check-in* adquiriendo únicamente aquellos que pertenecen a España, es decir, cuyo *country_code* tiene las siglas *ES,* como se puede ver en la Figura 3.2. Decidimos guardar los datos obtenidos en un *dataframe*, además de guardalos en un csv, para utilizarlos en los posteriores filtrados.

```
## Obtención de POIs en España a partir del ds de Foursquare WW
def filter_poi(line):
    data = line.strip().split('\t')

    row= {'Venue_ID': data[0],'Latitude': data[1],'Longitude': data[2],
          'Category': data[3], 'Country_Code':data[4]}
    return row

def read_poi(filename):
    """Obtiene los venues que se encuentran en España"""

    filepath = os.path.abspath(filename) #path absoluto

    with open(filepath, encoding="utf-8") as file:
        df = None
        df = pd.DataFrame([filter_venues(line) for line in file])
        df_es = df[df['Country_Code']=='ES'].reset_index(drop=True)

    return df_es
```

Figura 3.2: Implementación Python de la lectura y filtrado de primer nivel del fichero de localizaciones del dataset de Foursquare

A continuación, ejecutamos un segundo filtrado aplicando la función de *isin(category_list)* sobre el *dataframe* para extraer los POI pertenecientes a unas determinadas categorías que interesan en el proyecto mencionado anteriormente y recogidas en la Tabla 3.2.

Category	*Art Museum*
	Art Gallery
	American Restaurant
	Argentinian Restaurant
	Arepa Restaurant
	Brazilian Restaurant
	Burrito Place
	Cajun / Creole Restaurant
	Caribbean Restaurant
	Cuban Restaurant
	Latin American Restaurant
	Mexican Restaurant
	Monument / Landmark
	New American Restaurant
	Nightclub
	Nightlife Spot
	Other Nightlife
	Peruvian Restaurant
	Piano Bar
	South American Restaurant
	Taco Place

Tabla 3.2: Categorías seleccionadas que describen a los POI de nuestro dataset customizado

Finalmente, obtenemos los POI que se localizan en la Comunidad de Madrid a partir del geocodificador *Nominatim* de *GeoPy*. Aplicamos una geocodificación inversa empleando los valores de latitud y longitud para determinar la dirección completa del POI y poder filtrar aquellos que incluyan *'Comunidad de Madrid'*.

```
## Filtrado de coordenadas para Madrid
def filter_poi_mad(es_poi_df):
    geolocator = Nominatim(user_agent="datos_tfg")
    mad_df = es_poi_df.copy()
    l_indexes =[]
    for index, row in mad_df.iterrows():
        geoloc = '%s, %s' % (row['Latitude'],row['Longitude'])
        location = geolocator.reverse(geoloc)
        stripped = location.address.strip().split(', ')

        if 'Comunidad de Madrid' not in stripped:
            l_indexes.append(index)
    return mad_df
```

Figura 3.3: Implementación en Python del filtrado de POI ubicados en la Comunidad de Madrid

ii. **Adquirir *check-in***: empleamos el *dataframe* anterior en las funciones del filtrado de los *check-in* para poder descartar los que no se realizan en POI de Madrid. Al igual que en el filtrado de POI, guardamos estos datos tanto en un csv como en un dataframe por la misma razón, Figura 3.4.

iii. **Relaciones sociales de usuarios**: gracias al dataframe de *check-in* podemos extraer todos los usuarios que realizan estos *check-in* que nos importan para el filtrado del fichero de la red de amistades. Como podemos ver en la Figura 3.5, es importante realizar la comprobación de que ambos usuarios que forman parte de la relación que se puede obtener a partir del *dataframe* de *check-in*.

```
## Obtención de la Colección de checkins
def filter_WWW_Checkins(venues_ids,line):
    data = line.strip().split('\t')

    if data[1] in venues_ids:
        row= {'User_ID': data[0],'Venue_ID': data[1],
              'UTC_time': data[2],'Timezone_offset': data[3]}
        return row
    return None

def read_WWW_Checkins(venues_ids, filename):
    """Crea la Coleccion de checkins D del algoritmo"""
    filepath = os.path.abspath(filename)
    with open(filepath, encoding="utf-8") as file:
        data = []

        for line in file:
            row = filter_WWW_Checkins(venues_ids, line)
            if(row!=None):
                data.append(row)

        df_es = pd.DataFrame(data)

    return df_es
```

Figura 3.4: Implementación en Python de la lectura y filtrado de primer nivel del fichero de check-in del dataset de Foursquare

```
#Matriz de links sociales S
def filter_friendship(line, l_users):
    data = line.strip().split('\t')
    row = None
    if (data[0] in l_users) and (data[1] in l_users):
        row= {'User1_ID': data[0],'User2_ID': data[1]}
    return row

def read_friendship(l_users, filename):
    """ Obtiene los enlaces sociales que se
    encuentran entre usuarios en Madrid,España """

    filepath = os.path.abspath(filename)
    with open(filepath, encoding="utf-8") as file:
        data = []
        for line in file:
            row = filter_friendship(line, l_users)
            if row != None:
                data.append(row)
        df = pd.DataFrame(data)
    return df
```

Figura 3.5: Implementación en Python de la lectura y filtrado de primer nivel del fichero de relaciones sociales entre usuarios del dataset de Foursquare

3.2 Tecnologías

La reproducción del algoritmo LORE en el presente trabajo consta de dos tareas fundamentalmente: en primer lugar, la obtención del *dataset* personalizado para POI ubicados en Madrid y, en segundo lugar, la implementación del algoritmo LORE propiamente dicho. Tanto para realizar el filtrado del *dataset* original de Foursquare, como la implementación del algoritmo, hemos empleado el lenguaje **Python 3** en el entorno Jupyter, mediante el correspondiente desarrollo de *scripts* para la realización de ambas tareas.

En primer lugar, los datos originales del *dataset* se encuentran almacenados en formato de ficheros de texto. Tras el filtrado, el *dataset* personalizado a emplear en la evaluación se conserva en ficheros csv. En relación con la implementación del filtrado del *dataset* de Foursquare, se emplean varias librerías como clases Python:

- La librería *os*: se utiliza para localizar la ruta a los ficheros del *dataset*.

- La clase *Nominatim* de la librería *GeoPy*[2]: empleado en el filtrado de POI situados en Madrid. Por medio del método *reverse()* del geolocalizador *Nominatim* resolvemos la dirección de cada POI dados su latitud y su longitud.
- La librería *pandas*: esta librería se emplea para crear *dataframes* con los datos del *dataset* personalizado, cambiar el tipo de datos, realizar operaciones sobre los mismos durante el filtrado y para obtener estadísticas.

En segundo lugar, las librerías Python utilizadas para la implementación del algoritmo LORE son:

- La librería *pandas*: para acceder a los datos del *dataset* cuando se requiere y extraer los conjuntos de test y entrenamiento.
- La librería *datetime*: se utiliza el método *timedelta()* para calcular diferencias temporales entre dos *check-in* ya que, como podemos ver en la estructura del *dataset* de la Figura 3.1, la hora registrada de los *check-in* está en formato UTC y es necesario sumarle el offset de la zona horaria.
- La librería *math*: varios métodos se emplean en la estimación de los parámetros α, γ, β, η. Por ejemplo, el método *log()* interviene en el cálculo de cada parámetro y, los métodos *radians(), cos(), sin(), asin(), sqrt()* en el cálculo del semiverseno entre dos POI.
- El método *itemgetter()* de la librería *operator*: para ordenar el ranking final de POI recomendados.

3.3 LORE

Volviendo a la teoría de LORE, seguimos el pseudocódigo de la **Figura 3.6** para realizar la implementación en Python.

3.3.1 Consideraciones previas

Antes de empezar el desarrollo del algoritmo, es necesario explicar los parámetros de entrada que recibe la función implementada. Como podemos ver en la Figura 3.6, el algoritmo recibe como entrada la colección de *check-in D* y la matriz de enlaces sociales *S.* La colección *D* se define como un conjunto de *check-in* $\langle u_i, l_i, t_i \rangle$ para todos los usuarios

[2] https://geopy.readthedocs.io/en/stable/

$u_i \in U$ que han visitado todas las localizaciones $l_i \in L$ en el instante t_i, es decir, $D = \{\langle u_i, l_i, t_i \rangle\}_{i=1}^{|D|}$.

ALGORITHM 1: LORE: A gravity-model-based location recommender system

Input: Check-in collection D and social link matrix $\mathbf{S}_{|U|\times|U|}$.
Output: Top-k new locations for each user.

1: // **Phase 1: The data pre-processing phase**
2: Construct check-in frequency matrix $\mathbf{R}_{|U|\times|L|}$ from D
3: Construct location-location transition graph (L^2TG) from D (Section 3.1)
4: // **Phase 2: The parameter estimation phase**
5: Estimate $\alpha, \gamma, \beta, \eta$ based on Equations (9), (13), (20), (23), respectively
6: // **Phase 3: The visiting probability prediction phase**
7: **for** each $u \in U$ **do**
8: Extract check-in sequence for u from D: $L_u = \{\langle l_1, t_1 \rangle \rightarrow \langle l_2, t_2 \rangle \rightarrow \cdots \rightarrow \langle l_n, t_n \rangle\}$
9: Set user-specific latest check-in time: $t_c = t_n$
10: **for** each new location $l \in L \wedge l \notin L_u$ **do**
11: Initialize $\Pr(l|L_u) = 0$
12: **for** each visited location $l_i \in L_u$ **do**
13: // **Step 3.1: Distance derivation**
14: Compute temporal difference $x_{t_i,t_c} = t_c - t_i$ in Equation (5)
15: Compute spatial distance $y_{l_i,l} = Haversine(l_i, l)$ in Equation (6)
16: Compute $F_{Tem}(x_{t_i,t_c})$ based on Equation (10)
17: Compute $F_{Spa}(y_{l_i,l})$ based on Equation (14)
18: Compute $Distance(\langle l_i, t_i \rangle, \langle l, t_c \rangle) = 1/[F_{Tem}(x_{t_i,t_c}) \cdot F_{Spa}(y_{l_i,l})]$ in Equation (4)
19: // **Step 3.2: Mass derivation**
20: Compute $Mass(l_i) = \mathbf{R}_{u,l_i}$ in Equation (15)
21: Compute popularity p_l based on Equation (17)
22: Compute social check-in frequency $q_{u,l}$ based on Equation (18)
23: Compute $F_{Pop}(p_l)$ based on Equation (21)
24: Compute $F_{Soc}(q_{u,l})$ based on Equation (24)
25: Compute $Mass(l) = F_{Pop}(p_l) \cdot F_{Soc}(q_{u,l})$ in Equation (16)
26: // **Step 3.3: Weighing additive Markov chain with the gravity model**
27: Compute transition probability $TP(l_i \rightarrow l)$ based on Equation (1)
28: Compute $Gravity(\langle l_i, t_i \rangle, \langle l, t_c \rangle)$ based on Equation (3)
29: $\Pr(l|L_u) = \Pr(l|L_u) + TP(l_i \rightarrow l) \cdot Gravity(\langle l_i, t_i \rangle, \langle l, t_c \rangle)$ based on Equation (2)
30: **end for**
31: **end for**
32: **return** Top-k new locations with the highest visiting probability $\Pr(l|L_u)$ for u
33: **end for**

Figura 3.6: Pseudocódigo del algoritmo del modelo LORE extraído de [4]

En nuestra implementación, la colección *D* se identifica con dos de los ficheros del *dataset* personalizado que mantenemos en dos *dataframe*. Con respecto a la salida, nuestra implementación también recibe *k* como parámetro para indicar el ranking de POI que ha de devolver para cada usuario durante la evaluación.

3.3.2 Pre-procesado de datos

Comenzamos por la primera fase en la línea 1, en la que además de construir las estructuras expuestas en las líneas 2 y 3, construimos otras cuyos datos se requieren en líneas siguientes con el fin de optimizar el algoritmo.

En primer lugar, guardamos en un diccionario la secuencia espacio-temporal de *check-in* $L_u = \{\langle l_1, t_1 \rangle \rightarrow \langle l_2, t_2 \rangle \rightarrow \cdots \rightarrow \langle l_n, t_n \rangle\}$ para cada usuario *u* de la colección de

check-in D, siendo l_n el POI visitado y t_n el instante del *check-in*. La secuencia de *check-in* nos facilitará la creación del resto de estructuras.

```
## Obtencion de la Secuencia espacio-temporal de checkins L-sub-u
seq_st={}
U =  list(users['User_ID'])

for u in U:
    temp_df_chk = df_esCheckins[df_esCheckins['User_ID']==u]
    temp_df_chk = temp_df_chk.reset_index(drop=True)
    temp_df_chk.sort_values(by='UTC_time')

    seq_st[u] = {}
    for index, row in temp_df_chk.iterrows():
        seq_st[u][index] = {'Venue_ID': row['Venue_ID'],
                            'UTC_time': row['UTC_time'],
                            'Offset': row['Timezone_offset']}
```

Figura 3.7: Implementación en Python de la construcción de las secuencias de check-in por cada usuario de la colección *D*

Posteriormente, construimos en otro diccionario la matriz de frecuencias de *check-in R* de la línea 2. En nuestro código la matriz recibe el nombre *m_freq* como podemos ver en el *script* de la Figura 3.8.

```
## Matriz de frecuencias de checkins
m_freq = {}
L = list(mad_df['Venue_ID'])

for u in U:
    m_freq[u] = {}
    for l in L:
        l_freq = 0
        d_checks = seq_st[u]

        for i,li in d_checks.items():
            if l == li['Venue_ID']:
                l_freq+=1
        m_freq[u][l] = l_freq
```

Figura 3.8: Implementación en Python de la matriz de frecuencias de check-in *R*

Continuando con la línea siguiente, construimos el grafo de transición localización-localización $L^2TG = (L, E)$, con L conjunto de nodos y E conjunto de aristas. Para ello nos servimos de dos diccionarios distintos, uno almacena la información de $OCount(l_i)$ y el otro, de $TCount(l_i \rightarrow l_j).OCount(l_i)$ representa las transiciones del nodo l_i, un POI, a otra localización cualquiera. Es decir, indica para cuántos nodos es l_i el nodo predecesor. En cambio, $TCount(l_i \rightarrow l_j)$ mantiene la información de cada arista (l_i, l_j) que representa la

transición del POI l_i al POI l_j. Por lo tanto, presenta el número total de transiciones de un POI l_i a otro POI l_j. Estas transiciones existen para POI consecutivos, razón por la que empleamos la secuencia de *check-in* para actualizar el conteo.

```
#Construir gráfico L2TG
ocount = {}
# Inicialización
for l in L:
    ocount.update({l:0})
# Actualización
for u,l_c in seq_st.items():
    len_c = len(l_c.keys())
    if len_c>1:
        for i,v in l_c.items():
            if (v['Venue_ID'] in ocount.keys()) and i < (len_c-1):
                ocount[v['Venue_ID']] += 1
```

Figura 3.9: Implementación en Python de la estructura $\boldsymbol{OCount(l_i)}$ como parte del grafo $\boldsymbol{L^2TG}$

```
# Inicialización
tcount = {}
for li in l_venues:
    tcount[li] = {}
    for lj in l_venues:
        tcount[li][lj] = 0
# Actualización
for u,c in seq_st.items():
    len_c = len(c.keys())
    if len_c > 1:
        for i,v in c.items():
            if (v['Venue_ID'] in tcount.keys())  and (i < len_c-1):
                lj = c[i+1]['Venue_ID']
                if lj in tcount[v['Venue_ID']].keys():
                    tcount[v['Venue_ID']][lj] += 1
```

Figura 3.10: Implementación en Python de la estructura $\boldsymbol{TCount(l_i \rightarrow l_j)}$ como parte del grafo $\boldsymbol{L^2TG}$

Seguidamente, obtenemos la matriz de enlaces sociales $\boldsymbol{S}$ que guardamos en el diccionario de nombre *m_soc* como se puede ver en la Figura 3.11. Para construir la matriz nos basamos en la regla de que para dos usuarios distintos u y u', $S_{u,u'} = 1$ si existe una relación social entre ambos usuarios en la LSBN o, $S_{u,u'} = 0$ en caso contrario.

```
## Matriz de enlaces sociales
m_soc = {}
l_u1 = list(df_soc['User1_ID'])
l_u2 = list(df_soc['User2_ID'])
# Inicialización
for u1 in U:
    m_soc[u1] = {}
    for u2 in U:
        m_soc[u1][u2] = 0
# Actualización
for u in U:
    df_aux = None
    if u in l_u1:
        df_aux = df_soc[df_soc['User1_ID']==u]
        for i in list(df_aux['User2_ID']):
            m_soc[u][i] = 1
            m_soc[i][u] = 1
    if u in l_u2:
        df_aux = df_soc[df_soc['User2_ID']==u]
        for i in list(df_aux['User1_ID']):
            m_soc[u][i] = 1
            m_soc[i][u] = 1
```

Figura 3.11: Implementación en Python de la matriz de enlaces sociales *S*

3.3.3 Estimación de parámetros

El siguiente punto desarrolla la fase de estimación de los parámetros α, γ, β y η vista en la línea 5 del algoritmo de la Figura 3.6.

Comenzando por α, recuperamos una muestra *X* del conjunto *D* de diferencias temporales entre dos instantes de *check-in* consecutivos (t_j y t_{j+1}) de un usuario cualquiera, $X = \{t_{j+1} - t_j\}$. Adquirimos la muestra X desde la secuencia de *check-in* para usuarios con al menos 9 *check-in* con el fin de mejorar la estimación de α. Seguidamente, a partir de X calculamos una estimación de máxima similitud para hallar α dado por,

$$\boldsymbol{\alpha = 1 + |X| \left[\sum_{x \in X} \ln(x + 1) \right]^{-1}} \qquad \textbf{(3.1)}$$

En la Ecuación 3.1, $|X|$ es el tamaño de la muestra que está multiplicando a la inversa del sumatorio del logaritmo de cada valor de diferencia temporal de X.

Un proceso similar se sigue con γ. Es necesario obtener la muestra $Y = \{Haversine(l_j, l_{j+1})\}$ de D como un conjunto de distancias espaciales entre dos *check-in* consecutivos l_j y l_{j+1} de un usuario cualquiera. Para ello se emplea la función de *Haversine* o semiverseno que calcula la distancia a partir de la latitud y la longitud de los respectivos

POI donde se hace *check-in*. La implementación de Haversine empleada fue extraído de [22]. Al igual que para la muestra X, extraemos la muestra Y accediendo a los datos de la secuencia de *check-in* para usuarios con al menos 9 *check-in.* Ahora, con cada distancia *y* de la muestra *Y*, se puede estimar γ:

$$\boldsymbol{\gamma = 1 + |Y| \left[\sum_{y \in Y} \ln(y + 1) \right]^{-1}} \quad \textbf{(3.2)}$$

Como vemos en la Ecuación 3.2, $|Y|$ representa el tamaño de la muestra *Y* que está multiplicado por la inversa del sumatorio del logaritmo de cada distancia espacial *y* de *Y* más 1, como control cuando una distancia es 0.

Por otro lado, en la estimación de β únicamente intervienen el conjunto de POI *L* y la matriz de frecuencia de *check-in R*:

$$\boldsymbol{\beta = 1 + |L| \left[\sum_{l' \in L} \ln \left(\sum_{u' \in U} R_{u',l'} + 1 \right) \right]^{-1}} \quad \textbf{(3.3)}$$

La mayor dificultad de implementar la ecuación 3.3 recae en los dos sumatorios anidados. El tamaño del conjunto de POI , $|L|$, multiplica la inversa de: el sumatorio más externo por cada POI existente *l'* sobre el logaritmo del otro sumatorio, que recorre la matriz *R* por cada usuario *u'* y POI *l'* para obtener la frecuencia con la que *u'* visita *l'*.

Por último, en la estimación de η participan los conjuntos de usuarios *U* y de POI *L,* además de, las matrices de frecuencia *R* y de relaciones sociales *S:*

$$\boldsymbol{\eta = 1 + |U||L| \left[\sum_{u' \in U} \sum_{l' \in L} ln \left(\sum_{u'' \in U} S_{u',u''} \cdot R_{u'',l'} + 1 \right) \right]^{-1}} \quad \textbf{(3.4)}$$

En la Ecuación 3.4 tenemos que implementar tres sumatorios anidados. El tamaño de los conjuntos de usuarios y POI, $|U|$ y $|L|$, multiplican la inversa de: el sumatorio más externo para cada usuario *u'* de *U*, el interno siguiente por cada POI *l'* de *L* sobre el logaritmo del último sumatorio, que calcula la frecuencia de *check-in* de los amigos *u''* de *u'*.

Por último, introducimos una mejora para el rendimiento del algoritmo: el cálculo previo de la popularidad global de cada POI al no ser dependiente del usuario objetivo en

cada iteración y, que se obtiene en la línea 23 del algoritmo en la Figura 3.6. Guardamos cada valor en un diccionario y, dada la teoría, estimamos la distribución de popularidad de cada POI *l* sobre todos los usuarios del LBSN mediante,

$$\boldsymbol{F_{Pop}(p_l) = 1 - (p_l + 1)^{1-\beta}} \quad \textbf{(3.5)}$$

Llegados a este punto, $\boldsymbol{p_l}$ es el único parámetro de la ecuación 3.5 que falta por hallar. Para ello, se emplea la matriz de frecuencias *R* de manera que, $p_l = \sum_{u' \in U} R_{u',l}$.

3.3.4 Predicción de probabilidad

En esta fase nos encontramos con tres sub-fases en el algoritmo de la Figura 3.6. La primera fase se centra en la derivación de la distancia entre dos POI, le sigue la derivación de la masa de cada POI y, cómo última fase, se calcula la fuerza gravitatoria y la probabilidad de que un usuario visite un POI no visitado por medio de la cadena de Markov de adición.

Comenzando por la derivación de la distancia entre el POI visitado l_i y el no visitado l, se sigue la ecuación de la teoría de LORE $\frac{1}{Distancia(\langle l_i,t_i\rangle,\langle l,t_c\rangle)} = F_{Tem}(x_{t_i,t_c}) \cdot F_{Spa}(y_{l_i,l})$, donde $F_{Tem}(x_{t_i,t_c})$ es la distribución de la diferencia temporal y, $F_{Spa}(y_{l_i,l})$ es la distancia espacial. Por lo tanto, hallar la distancia se reduce a calcular:

$$\boldsymbol{Distancia(\langle l_i,t_i\rangle,\langle l,t_c\rangle) = \frac{1}{F_{Tem}(x_{t_i,t_c}) \cdot F_{Spa}(y_{l_i,l})}} \quad \textbf{(3.6)}$$

En otras palabras, la distancia entre dos POI es inversamente proporcional a la distribución de la diferencia temporal y la distancia espacial entre ambos puntos. Tanto $F_{Tem}(x_{t_i,t_c})$ como $F_{Spa}(y_{l_i,l})$ son funciones decrecientes. Esto significa que la distribución de la diferencia temporal disminuye cuanto mayor tiempo transcurre entre dos POI, reflejado en su fórmula

$$\boldsymbol{F_{Tem}(x_{t_i,t_c}) = (x_{t_i,t_c} + 1)^{1-\alpha}} \quad \textbf{(3.7)}$$

, con $x_{t_i,t_c} = t_i - t_c$ siendo la diferencia temporal entre el instante de *check-in* del POI visitado t_i y, del POI no visitado t_c. El valor del instante t_c se fija al check-in del último POI visitado, t_n, del historial del usuario objetivo. Una situación similar ocurre con la distribución

de la distancia espacial que disminuye cuanto mayor es la distancia espacial entre ambos POI. Esto se aprecia en su fórmula siguiente,

$$\boldsymbol{F_{Spa}(y_{l_i,l}) = \left(y_{l_i,l} + 1\right)^{1-\gamma}} \quad \textbf{(3.8)}$$

, donde $y_{l_i,l}$ es la diferencia espacial entre la ubicación del POI visitado l_i y el no visitado l y, se puede hallar mediante la función Haversine, $y_{l_i,l} = Haversine(l_i, l)$. Durante la estimación de γ, correspondiente a la ecuación 3.2, vimos que la función Haversine o semiverseno nos permite obtener la distancia entre dos puntos mediante su longitud y su latitud.

Prosiguiendo con la derivación de la masa, debemos de calcular tanto la masa del POI no visitado, *l,* como el del POI visitado, l_i, dado el usuario objetivo *u*. La masa del POI visitado se puede hallar directamente por medio de la matriz de frecuencias *R* con $Masa(l_i) = R_{u,l_i}$. En cambio, para calcular la masa del POI no visitado es necesario utilizar la popularidad del POI entre todos los usuarios y no solo entre sus amigos. La razón de ello parte de la dispersión de los datos de *check-in* que se muestran en la Tabla 4.1, ya que los usuarios han visitado una proporción de POI pequeña. Por ello para cuantificar su valor se reduce a:

$$\boldsymbol{Masa(l) = F_{Pop}(p_l) \cdot F_{Soc}(q_{u,l})} \quad \textbf{(3.9)}$$

Tanto $F_{Pop}(p_l)$ como $F_{Soc}(q_{u,l})$ son funciones crecientes que modelan las relaciones directamente proporcionales de la fuerza de atracción con la popularidad y la influencia social, respectivamente. Empezando por la estimación de la distribución de popularidad $F_{Pop}(p_l)$, su valor para cada POI se calculó previamente en la fase *Estimación de parámetros* por medio de la ecuación 3.5, luego, solo es necesario acceder al valor del diccionario para el POI no visitado.

Acerca de la distribución de la frecuencia social de *check-in*, se estima mediante la ecuación 3.10:

$$\boldsymbol{F_{Soc}(q_{u,l}) = 1 - \left(q_{u,l} + 1\right)^{1-\eta}} \quad \textbf{(3.10)}$$

, donde $q_{u,l} = \sum_{u' \in U} S_{u,u'} R_{u',l}$ con $S_{u,u'}$ indicando la existencia de una relación de amistad entre el usuario *u* y *u'* y $R_{u',l}$, la frecuencia con la que el usuario *u* hace *check-in* en el POI *l.*

En este punto del algoritmo, introducimos la funcionalidad para el control de los inicios en frío cuando un usuario no ha establecido ninguna relación de amistad. Debido a Esta situación provoca que $F_{Soc} = 0$ y, por lo tanto, $Masa(l) = 0$. Debido a ello, establecemos la regla siguiente: si $F_{Soc} = 0$, entonces $F_{Soc} = 1$. De esta forma, la masa del POI no visitado pasará a depender solamente de la popularidad global del POI.

Llegamos a la última línea de esta fase donde finalmente, calculamos la probabilidad de que un usuario *u* visite un POI no visitado aplicando la cadena de Markov n-ésima de adición de la ecuación 3.11.

$$\boldsymbol{Pr\ (l|L_u) = \sum_{i=0}^{n} TP(l_i \rightarrow l)\ \cdot\ Gravedad(\langle l_i, t_i\rangle, \langle l, t_c\rangle)} \qquad \textbf{(3.11)}$$

Como vemos en la Ecuación 3.11, en la cadena de Markov n-ésima la probabilidad de transición $TP(l_i \rightarrow l)$ se pondera por la fuerza de atracción entre un POI visitado y uno no visitado, dados por el modelo de gravedad propuesto,

$$\boldsymbol{Gravedad(\langle l_i, t_i\rangle, \langle l, t_c\rangle) = \frac{Masa(l_i)\ \cdot\ Masa(l)}{Distancia(\langle l_i, t_i\rangle, \langle l, t_c\rangle)}} \qquad \textbf{(3.12)}$$

La gravedad se puede calcular directamente pues en este punto, todos sus componentes han sido previamente calculados en secciones anteriores a esta fase. Mientras tanto, la probabilidad de transición, si $OCount(l_i) > 0$, viene dado por:

$$\boldsymbol{TP(l_i \rightarrow l_j) = \frac{TCount(l_i \rightarrow l_j)}{OCount(l_i)}} \qquad \textbf{(3.13)}$$

En caso contrario, se dan dos situaciones: $TP(l_i \rightarrow l_j) = 1$, si l_i y l son el mismo POI o, $TP(l_i \rightarrow l_j) = 0$ si son distintos POIs. En el algoritmo solo se contempla la segunda situación ya que, como se puede ver en las líneas 10 y 12 de la Figura 3.6, los bucles se realizan sobre dos conjuntos de POI distintos.

Finalmente, podemos estimar la probabilidad del POI no visitado por medio de la ecuación 3.11. Cada valor es almacenado en un diccionario y tras todas las iteraciones, se ordenan de forma descendente a fin de devolver el top-k POI con mayor probabilidad de visita $Pr\ (l|L_u)$ para el usuario *u*.

4 Resultados

En este capítulo se presentan las pruebas y los resultados de la evaluación de la reproducción del algoritmo LORE [4] aplicando nuestro *dataset* personalizado, así como dificultades encontradas en el proceso.

4.1 Aspectos previos

El primer aspecto a tener en cuenta es la división de los datos del *dataset*. En una primera iteración, realizamos la división en términos de *check-in* en un conjunto de entrenamiento y un conjunto de prueba con una proporción 70-30. Para ello, seguimos la idea descrita por Zhang et al.: ordenamos el conjunto de *check-in* desde los más antiguos a los más recientes de manera que durante la división, los *check-in* más antiguos componen el conjunto de entrenamiento y los más recientes, el conjunto de test.

Otro aspecto a decidir para examinar la calidad de LORE es el número de POI a recomendar, es decir, el valor de *k*. Decidimos realizar recomendaciones de 3, 5 y 7 POI máximo ya que un mayor número de recomendaciones resulta innecesario para los usuarios.

En cuanto a las métricas utilizadas para la evaluación de la calidad del algoritmo, utilizamos la precisión y el recall, definidos en [4] como:

$$\textbf{Precisión} = \frac{\textbf{\# de POIs descubiertos}}{\textbf{\# POIs a recomendar al usuario por el sistema: } \boldsymbol{k}} \qquad \textbf{(4.1)}$$

$$\textbf{Recall} = \frac{\textbf{\# de POIs descubiertos}}{\textbf{\# POIs visitados realmente por el usuario en el cjto. test}} \qquad \textbf{(4.2)}$$

Decidimos aplicar las fórmulas a la recomendación de cada usuario y posteriormente, calculamos la media aritmética de entre todos los resultados. El script implementado determina la precisión y el recall simultáneamente como se puede ver en la Figura 4.1.

```python
def precision_recall_test_set(dataframe, k, checkins_test):
    precision = {}
    recall = {}
    for i,r in dataframe.iterrows():

        aux = checkins_test[checkins_test['User_ID']==r['User_ID']]
        count_pois = 0

        for index,row in aux.iterrows():
            for i in range(1,k+1):

                if aux[aux['Venue_ID']== r['Top_%s' % i]].empty == False:

                    count_pois += 1
        precision[r['User_ID']] = count_pois/k
        if len(list(aux['Venue_ID'])) > 0:
            recall[r['User_ID']] = count_pois/len(list(aux['Venue_ID']))
        else: recall[r['User_ID']] = 0.0

    return precision,recall
```

Figura 4.1: Implementación en Python de la función que realiza el cálculo de las métricas precisión y recall para la evaluación de LORE

La función recibe como parámetros de entrada: el *dataframe* con el *top-k* POI recomendados a cada usuario, el valor *k* de POI recomendados y, el *dataframe* de *check-in* del conjunto de test. Por cada usuario que reconocemos por su identificador (*r['User_ID']* en el código), extraemos sus *check-in* del conjunto de test. A continuación, comprobamos si algún *check-in* del conjunto de test se encuentra en el ranking y se incrementa el contador cada vez que se encuentra una coincidencia. Con el valor de *k* podemos comprobar dinámicamente si algún *check-in* de testeo se encuentra en cada *top-k* recomendaciones. En el código, *r['Top_%s' % i]* accede a cada valor del ranking. Por último, se calculan los valores de precisión y recall, además de controlar valores nulos del denominador para el recall.

4.2 Pruebas y resultados

4.2.1 Pruebas durante la evaluación

En unas primeras pruebas del funcionamiento del algoritmo nos encontramos que se generaba una proporción altamente reducida de recomendaciones debido a la popularidad de los POI no visitados entre la red de amistades de estos usuarios. Por dicha razón, introdujimos un control de comienzo en frío previamente descrita en la sección *3.3.4 Predicción de probabilidad*, que resolvió el problema.

Dentro de este marco, realizamos una comparación entre la densidad de la matriz de frecuencia de *check-in* de nuestro *dataset* personalizado, que se identifica en [4] con la matriz usuarios-POI y, la versión del *dataset* de Foursquare global empleado en [4] como se puede ver en la Tabla 4.1. Se puede apreciar como la densidad de nuestro *dataset* es ligeramente mayor, sin embargo, la proporción de *check-in* con respecto al número de usuarios y POI revela el mayor historial de visitas por parte de los usuarios. Esto provoca una desventaja en nuestra implementación ya que el algoritmo aprende el comportamiento de un usuario analizando al completo las secuencias de sus visitas.

	Foursquare (Madrid)	Foursquare (Global)
Número de POI	1,027	182,968
Número de check-in	2,551	1,385,223
Número de usuarios	828	11,326
Número de relaciones sociales	1,983	47,164
Densidad usuarios-poi	2.39×10^{-3}	2.3×10^{-4}

Tabla 4.1: Estadísticas del dataset personalizado y el original de Foursquare utilizado en [4]

La diferencia en densidad del historial de *check-in* se manifiesta durante la *Estimación de parámetros* en la sección 3.3.3. Los resultados obtenidos para los parámetros α, γ, β, η entran en un rango aceptable ya que dadas las ecuaciones 3.5, 3.7, 3.8 y 3.10 tenemos que $(1-\beta), (1-\alpha), (1-\gamma), (1-\eta) < 0$, cumpliendo con la teoría de Zhang et al [4]. Los valores de cada parámetro son los siguientes:

- $\alpha \approx 1.1858$: tiene sentido en comparación con el resultado para el *dataset* original dado que depende de los instantes de *check-in* y estos no son abundantes en general y, pueden estar o muy concentrados durante un periodo de tiempo o muy dispersos en el tiempo.
- $\gamma \approx 2.0701$: el resultado obtenido tiene sentido suponiendo una mayor distancia entre los POI consecutivos visitados.
- $\beta \approx 1.1346$: al igual que con α, el valor es menor en comparación con el obtenido en el *dataset* original. En este caso, al reducirse la cantidad de *check-ins* la suma de las frecuencias disminuye también.
- $\eta \approx 1.0003$: se obtiene un resultado similar a α y β pero en este caso la diferencia es más notable. Esto se debe a la cantidad de enlaces sociales. En nuestro *dataset* los enlaces son solo el doble del número de usuarios totales, mientras que en el *dataset* de Zhang et al. son el cuádruple de los usuarios.

En cuanto a la evaluación global del algoritmo LORE empleando el *dataset* personalizado, la primera iteración resultó fallida durante el cálculo de las métricas. Tanto la precisión como el recall devolvían valores nulos para los tres casos de recomendaciones.

En primer lugar, comprobamos el correcto funcionamiento del método implementado para calcular las métricas proponiendo un escenario con distintos rankings sobre los que calcular la precisión y recall.

A continuación, analizamos los check-in de entrenamiento, de test y los originales del dataset personalizado en busca de la causa por la que no obtenemos ninguna recomendación acertada. Llegamos a la conclusión de que el problema se encuentra en la división de los check-in en el conjunto test y de entrenamiento. El planteamiento que seguimos es el establecido por Zhang et al. [4] aunque con proporción 70-30, "el 80% de los datos de check-in con timestamps más antiguos se usan como el conjunto de entrenamiento y el otro 20% de datos de check-in se usan como el conjunto de test". Esta descripción deja abierta la posibilidad de realizar la división sobre el conjunto de check-in global o, sobre los check-in más antiguos del historial de cada usuario, siendo el primero el que elegimos durante la primera iteración. Como se puede ver en la Figura 4.2, en nuestro dataset personalizado existe un gran número de usuarios con menos de 5 *check-in* en su historial, seguido del rango de 5 a 10 *check-in*. Desconociendo el timestamp de cada *check-in*, puede darse que un gran número de ellos o todo el historial de un mismo usuario se encuentre entre los más antiguos del conjunto total. Esto queda expuesto tras comprobar cuantos usuarios del conjunto de entrenamiento se encuentran en el test: de los 404 usuarios del conjunto de test, solo hay 221 coincidencias con los 645 usuarios que forman parte del entrenamiento. La proporción de coincidencias no llega a la mitad de los usuarios que forman parte del entrenamiento y solo es ligeramente superior a la mitad de pertenecientes al test. Por ello, consideramos ser una de las causas de no obtener recomendaciones de calidad.

Para la siguiente iteración decidimos realizar la división sobre el historial de visitas de cada usuario. Aplicamos la división sobre las secuencias de *check-in* de cada usuario, de manera que, se obtienen los más antiguos para el conjunto de entrenamiento y nuevamente con una proporción 70-30. El resultado en el conjunto global de *check-in* del *dataset* se traduce a un 73% para el conjunto de entrenamiento, incluyendo 828 usuarios y, un 27% para el test, con 377 usuarios. A continuación, analizamos ambos conjuntos, cuyos resultados se muestran en la Figura 4.3 (b), y observamos que los 377 usuarios del test se encuentran también en el conjunto de entrenamiento. Gracias a esta división, conseguimos solucionar el problema de realizar recomendaciones con LORE que puedan ser evaluados en precisión y recall.

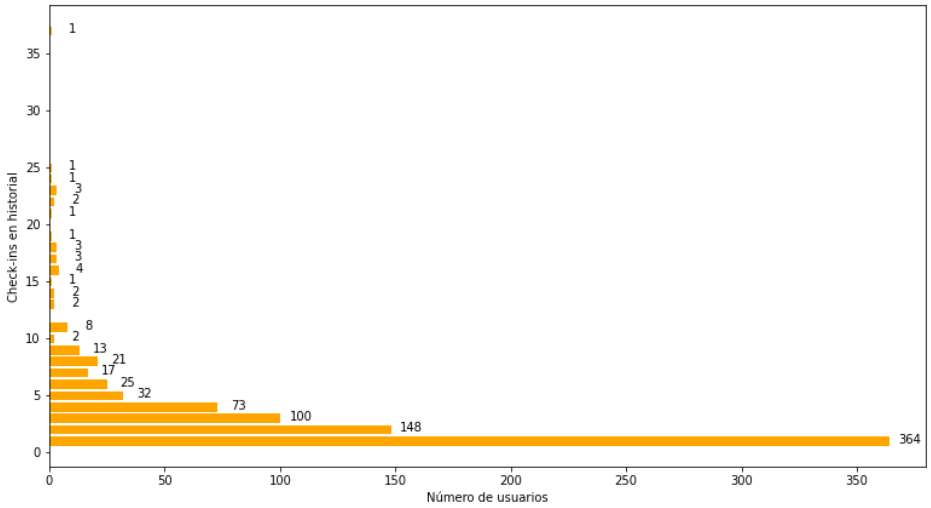
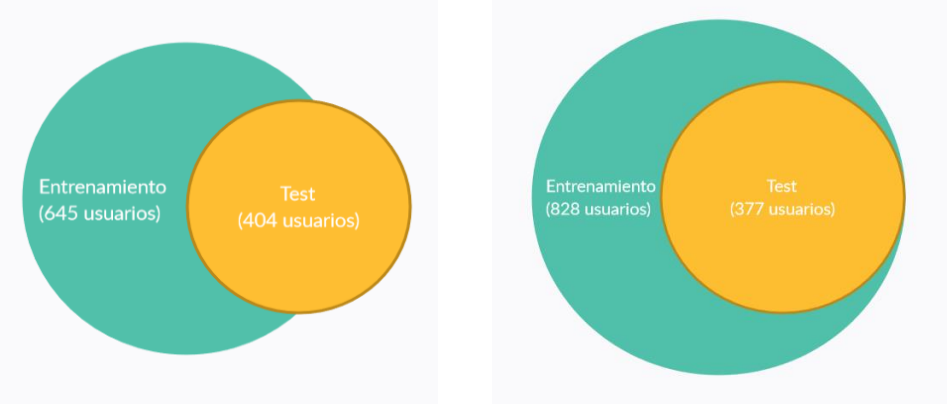


Figura 4.2: Gráfica de frecuencia de número de check-ins en el historial de los usuarios de nuestro dataset personalizado



(a) División en la primera iteración **(b)** División en la segunda iteración

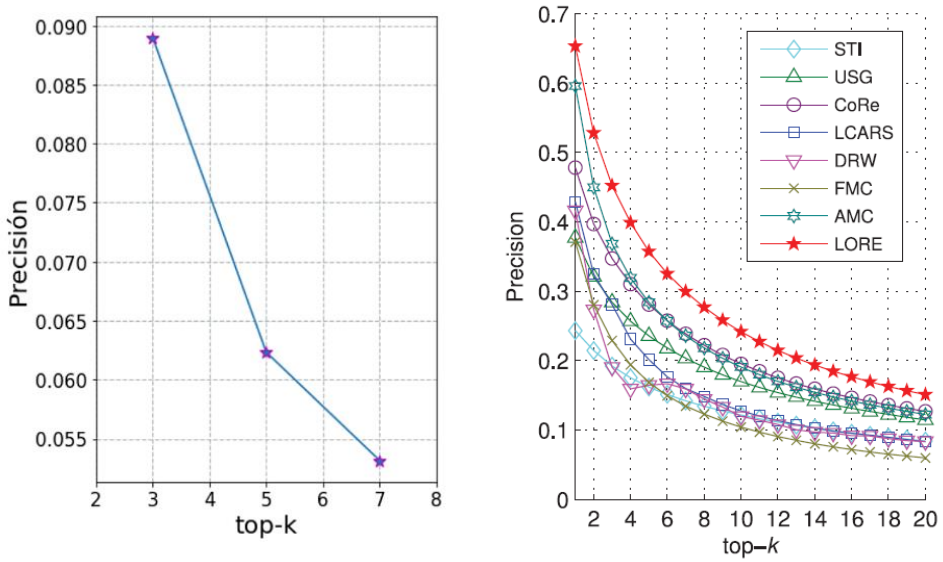
Figura 4.3: Coincidencia de usuarios entre los conjuntos de entrenamiento y de test

4.2.2 Comparación de resultados y discusión

En las Figuras 4.4 y 4.5 comparamos la exactitud de las recomendaciones con respecto el número de POI recomendados (*top-k*) por el algoritmo reproducido de LORE y la obtenida en [4]. Es necesario resaltar que la exactitud de las recomendaciones de POI suele ser baja debido a la densidad de los *dataset* que es menor en comparación con los *dataset* utilizados en SR convencionales. Por ejemplo, la densidad del *dataset* de

Movielens10M es 1.34% mientras que el del global de Foursquare es 0.0034% [26]. Además, se reporta una precisión máxima de 0.06 sobre un *dataset* de 2.72×10^{-4} de densidad en [27] y, 0.03 sobre dos *dataset* con densidades de 9.85×10^{-4} and 6.35×10^{-3} en [28]. Particularmente, en nuestra evaluación la precisión no supera el 0.09 y el recall el 0.15. Como se esperaba, estos valores aparecen debido al tamaño de nuestro *dataset* personalizado y la cantidad de datos de *check-in*.

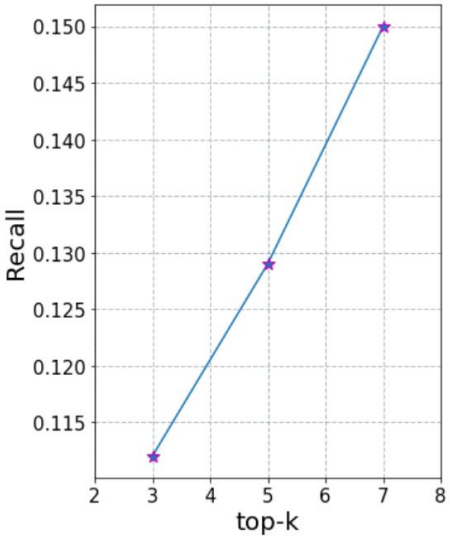
Otro aspecto a señalar es el número de POI recomendados para evaluar nuestra implementación. Realizar recomendaciones de un mayor número de POI no resulta de ayuda para los usuarios. Al igual que en los resultados de Zhang et al. para LORE en [4], a medida que el valor de *k* incrementa, el recall aumenta, pero la precisión va disminuyendo, reflejado en las Figuras 4.4 y 4.5. En general, al recomendar un mayor número de POI a usuarios, son capaces de descubrir más lugares de los que en principio desearían visitar. No obstante, estos POI tienen una menor posibilidad de que a los usuarios les guste por sus bajas probabilidades de visita en el ranking devuelto por LORE.



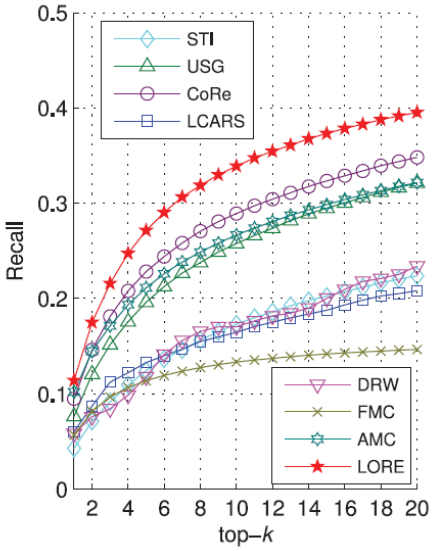
(a) Precisión sobre nuestro dataset personalizado

(b) Precisión sobre Foursquare extraído de [4]

Figura 4.4: Precisión de las recomendaciones con respecto a los *k* POI recomendados a usuarios



(a) Recall sobre nuestro dataset personalizado



(b) Recall sobre Foursquare extraído de [4]

Figura 4.5: Recall de las recomendaciones con respecto a los *k* POI recomendados a usuarios

5 Conclusiones y trabajo futuro

En este artículo, hemos reproducido un modelo de recomendación de POI basado en la Gravedad conocido como LORE. Este modelo explota los *check-in* secuenciales del historial del usuario para cuantifica el efecto que tiene una localización visitada sobre otra no visitada gracias al modelo gravitatorio que integra los aspectos espacio-temporales, sociales y la popularidad. Asimismo, emplea una cadena de Markov de orden n-ésima para determinar con mayor efectividad la probabilidad de que un usuario visite un POI.

Enfocamos este algoritmo sobre un caso particular, un *dataset* con POI pertenecientes a la Comunidad de Madrid para estudiar la efectividad de su comportamiento en zonas puntuales. Los resultados experimentales obtenidos sobre esta colección demuestran la escasez de *datasets* completos disponibles para recomendación de POI. En particular, con un número reducido de datos y, en concreto, de *check-in* registrados en zonas centralizadas no pertenecientes a ciudades metropolitanas donde se concentra la mayor actividad. A causa de ello, la exactitud de las recomendaciones varía considerando una o múltiples países o ciudades, ya que los datos registrados no son constantes en todas las áreas. Siendo dicho escenario el problema más destacable en los resultados obtenidos durante nuestra evaluación de la reproducibilidad del algoritmo LORE.

En cuanto al trabajo futuro, dado que no es una tarea fácil recopilar datos en masa que componen un *dataset* y, vista la influencia que tiene la densidad de los mismos para proporcionar una recomendación de calidad, proponemos la creación de un *dataset* con POI en la Comunidad de Madrid como un proyecto particular a vista de mejorar la recomendación del algoritmo reproducido en este TFG.

Referencias

[1] J. Schafer, J. Konstan and J. Riedl, Data Mining and Knowledge Discovery, vol. 5, no. 12, pp. 115-153, 2001. Available: 10.1023/a:1009804230409

[2] M. Sigala, "Destination Recommendation Systems: Behavioural Foundations and Applications20082D.R. Fesenmaier, H. Werthner and K.W. Wober (Eds). Destination Recommendation Systems: Behavioural Foundations and Applications. Wallingford, Oxfordshire: Cabi Publishing 2006. 347 pp., ISBN: -10: 0-85199-023-1 US$100.00", *International Journal of Contemporary Hospitality Management*, vol. 20, no. 2, pp. 236-237, 2008. Available: 10.1108/09596110810852212

[3] J. Bao, Y. Zheng, D. Wilkie and M. Mokbel, "Recommendations in location-based social networks: a survey", *GeoInformatica*, vol. 19, no. 3, pp. 525-565, 2015. Available: 10.1007/s10707-014-0220-8

[4] J. Zhang and C. Chow, "Spatiotemporal Sequential Influence Modeling for Location Recommendations", *ACM Transactions on Intelligent Systems and Technology*, vol. 7, no. 1, pp. 1-25, 2015. Available: 10.1145/2786761

[5] R. Burke, *User Modeling and User-Adapted Interaction*, vol. 12, no. 4, pp. 331-370, 2002. Available: 10.1023/a:1021240730564

[6] R. Burke, "Hybrid Web Recommender Systems", *The Adaptive Web*, pp. 377-408. Available: 10.1007/978-3-540-72079-9

[7] J. Lu, D. Wu, M. Mao, W. Wang and G. Zhang, "Recommender system application developments: A survey", *Decision Support Systems*, vol. 74, pp. 12-32, 2015. Available: 10.1016/j.dss.2015.03.008

[8] Zhao, S., King, I., & Lyu, M. R. "A survey of point-of-interest recommendation in location-based social networks.", 2016. *arXiv preprint arXiv:1607.00647*

[9] R. Baral and T. Li, "Exploiting the roles of aspects in personalized POI recommender systems", *Data Mining and Knowledge Discovery*, vol. 32, no. 2, pp. 320-343, 2017. Available: 10.1007/s10618-017-0537-7

[10] Y. Liu, T. Pham, G. Cong and Q. Yuan, "An experimental evaluation of point-of-interest recommendation in location-based social networks", *Proceedings of the VLDB Endowment*, vol. 10, no. 10, pp. 1010-1021, 2017. Available: 10.14778/3115404.3115407

[11] Y. Koren, R. Bell and C. Volinsky, "Matrix Factorization Techniques for Recommender Systems," in Computer, vol. 42, no. 8, pp. 30-37, Aug. 2009, doi: 10.1109/MC.2009.263.

[12] X. Li, G. Cong, X. Li, T. Pham and S. Krishnaswamy, "Rank-GeoFM", *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2015. Available: 10.1145/2766462.2767722

[13] H. Li, Y. Ge, R. Hong and H. Zhu, "Point-of-Interest Recommendations", *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016. Available: 10.1145/2939672.2939767

[14] H. Ma, C. Liu, I. King and M. Lyu, "Probabilistic factor models for web site recommendation", *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information - SIGIR '11*, 2011. Available: 10.1145/2009916.2009955

[15] C. Cheng, H. Yang, I. King, and M. R. Lyu. “Fused matrix factorization with geographical and social influence in location-based social networks” *In AAAI*, 2012.

[16] B. Liu, H. Xiong, S. Papadimitriou, Y. Fu and Z. Yao, "A General Geographical Probabilistic Factor Model for Point of Interest Recommendation", *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, no. 5, pp. 1167-1179, 2015. Available: 10.1109/tkde.2014.2362525

[17] M. Ye, P. Yin, W. Lee and D. Lee, "Exploiting geographical influence for collaborative point-of-interest recommendation", *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information - SIGIR '11*, 2011. Available: 10.1145/2009916.2009962

[18] T. Kurashima, T. Iwata, G. Irie and K. Fujimura, "Travel route recommendation using geotags in photo sharing sites", *Proceedings of the 19th ACM international conference on Information and knowledge management - CIKM '10*, 2010. Available: 10.1145/1871437.1871513

[19] Cheng, C., Yang, H., Lyu, M. R., & King, I. “Where you like to go next: successive point-of-interest recommendation.” In *Proceedings of the Twenty-Third international joint conference on Artificial Intelligence* (*IJCAI '13*). AAAI Press, 2605–2611. (2013, June).

[20] Z. Chen, H. Shen and X. Zhou, "Discovering popular routes from trajectories", *2011 IEEE 27th International Conference on Data Engineering*, 2011. Available: 10.1109/icde.2011.5767890

[21] W. Sinnott, *Virtues of the haversine*', Sky and Telescope, Vol.68, No.2, p.159, August, 1984.

[22] H. points), M. Dunn and S. Malyutin, "Haversine Formula in Python (Bearing and Distance between two GPS points)", *Stack Overflow*, 2021. [Online]. Available: https://stackoverflow.com/questions/4913349/haversine-formula-in-python-bearing-and-distance-between-two-gps-points. [Accessed: 17- May- 2021]

[23] D. Yang, B. Qu, J. Yang and P. Cudre-Mauroux, "LBSN2Vec++: Heterogeneous Hypergraph Embedding for Location-Based Social Networks", *IEEE Transactions on Knowledge and Data Engineering*, pp. 1-1, 2020. Available: 10.1109/tkde.2020.2997869

[24] E. Cho, S. Myers and J. Leskovec, "Friendship and mobility", *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '11*, 2011. Available: 10.1145/2020408.2020579

[25] "Yelp Dataset", *Yelp.com*, 2021. [Online]. Available: https://www.yelp.com/dataset. [Accessed: 17- January- 2021].

[26] P. Sánchez and A. Bellogín, "Applying reranking strategies to route recommendation using sequence-aware evaluation", *User Modeling and User-Adapted Interaction*, vol. 30, no. 4, pp. 659-725, 2020. Available: 10.1007/s11257-020-09258-4

[27] M. Ye, P. Yin, W. Lee and D. Lee, "Exploiting geographical influence for collaborative point-of-interest recommendation", *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information - SIGIR '11*, 2011. Available: 10.1145/2009916.2009962.

[28] Q. Yuan, G. Cong, Z. Ma, A. Sun and N. Thalmann, "Time-aware point-of-interest recommendation", *Proceedings of the 36th international ACM SIGIR conference on Research and development in information retrieval*, 2013. Available: 10.1145/2484028.2484030.